



Technische Hochschule Ingolstadt

Seminararbeit/Whitepaper

Umgehen von USB-Deskriptor basierten USB-Richtlinien am Beispiel einer Virtual Desktop Infrastruktur Umgebung

angefertigt von

Name: Dominik Gunther Florian Schlecht
Matrikelnummer: 00032209

Betreuer:

Technische Hochschule Ingolstadt: Dr. Paul Stanislaus
Allianz Deutschland AG: Dr. Ulrich Stremmel und Stephan

Ingolstadt, 17. Januar 2015

Inhaltsverzeichnis

1. Einleitung	1
2. Szenario	2
3. Universal Serial Bus	4
3.1. Gefahren bei USB	4
3.1.1. Schadsoftware	4
3.1.2. Datenabfluss	5
3.1.3. Exploits auf Treiberebene	5
3.2. Deskriptoren	6
4. Richtlinien	7
5. Umgehung der USB-Richtlinie	8
5.1. Wie wird gefiltert?	8
5.2. Teensy	8
5.3. Konzept	9
5.4. Proof of Concept	12
6. Fazit und Gegenmaßnahmen	13
A. Appendix	14
A.1. Quellcode	14
A.2. Ergänzende Grafiken	15
A.3. Quellcode Grafiken	15

1. Einleitung

In Zeiten von *Heartbleed*[3] und *Shellshock*[9], der *Snowden-Leaks* und der *NSA-Affäre*[10] und der fortlaufenden Digitalisierung der Industrie und Gesellschaft wird das Thema Informationssicherheit immer wichtiger. Daten werden, unabhängig davon, ob diese Privatpersonen oder Unternehmen zugeordnet sind, immer wertvoller. So ergeben sich beispielsweise aus einem gehackten Smartphone einer Privatperson Informationen wie E-Mail-Adressen, Kontakte und Chat-Verläufe bis hin zu Passwörtern für Online-Banking oder persönlichen Bildern. Wenn diese Informationen auf dem Schwarzmarkt verkauft oder online veröffentlicht werden, kann dies für die Personen oft Reputations- wie auch finanzielle Schäden nach sich ziehen. Diese Tätigkeiten werden unter anderem oft unter dem Schlagwort „Cybercrime“ zusammengefasst. Betrachtet man Unternehmen, so ist der mögliche finanzielle Schaden wesentlich höher als für Privatpersonen. Durch die Entwendung von Kreditkartendaten erlitten zum Beispiel mehrere Supermärkte in den USA beträchtliche Reputationsschäden [1][4]. Eventuell noch höhere Schäden könnte es nach sich ziehen, wenn streng vertrauliche Dokumente von Unternehmen, wie z.B. Konstruktionsskizzen für ein neues Automodell, Quellcode oder vorläufige Geschäftsberichte durch Hacker erbeutet und an ein Konkurrenzunternehmen verkauft würden. Dies hört sich unreal an, aber die Firma McAfee schätzt den Verlust für die Wirtschaft durch „Cybercrime“ im Jahr 2014 auf bis zu 575 Milliarden USD[5]. Um diesem Trend entgegen zu wirken, müssen Unternehmen Maßnahmen ergreifen, welche das Schutzniveau erhöhen. Oft werden hier aufgrund der technischen Sicht nur im Internet erreichbare Komponenten beachtet, wie das schnelle Patching von Servern. Dies ist in Hinsicht auf *Poodle*[8] und *Shellshock*[9] sicherlich auch notwendig, jedoch sollte man alle Wege, über welche Daten von Dritten in das Unternehmen gelangen, Daten an Dritte weitergegeben werden könnten, und alle internen Bedrohungen wahrnehmen, einschätzen und eindämmen. Eine solche Prüfung war die Grundlage für dieses Dokument.

2. Szenario

Bei einer Prüfung interner Regularien bei der Informationssicherheit wurde das Thema USB-Geräte in Verbindung mit Thin-Clients aufgegriffen. Hier soll aus gegebenen fachlichen Anlässen eine Möglichkeit geschaffen werden, die Daten lokaler USB-Geräte, wie z.B. USB-Sticks oder USB-CD-Laufwerke, an die virtuelle Maschine des Benutzers weiter zu leiten. In dieser Arbeit galt es, das Risiko zu prüfen und entsprechende Gegenmaßnahmen zu entwickeln. Würde man das Durchstellen von USB-Geräten im Allgemeinen erlauben, so würden sich erhebliche Gefahren ergeben, welche unter 3.1 erläutert werden. Um dem vorzubeugen, soll auf Basis einer Richtlinie, welche in 4 weiter erläutert wird, das Durchstellen auf bestimmte Geräte begrenzt werden. Dies geschieht bei dem hier getesteten Produkt über die Filterung nach USB-Deskriptoren wie *idVendor* und *idProduct*, welche ein USB-Device, z.B. einen bestimmten USB-Stick oder ein USB-CD-Laufwerk, eindeutig identifizieren sollen. Diese Felder werden unter 3.2 weiter erläutert. Bei der durchgeführten Sicherheitsprüfung stellte sich jedoch heraus, dass USB-Deskriptoren keinen Sicherungen unterliegen und somit mit Hilfe bestimmter Geräte gezielt emuliert werden können. Diese Umgehung der Sicherheits-Policy soll in diesem Dokument erläutert und aufgezeigt werden. Der Proof-Of-Concept findet sich unter 5.4.

Schritt 1: Ein USB-Gerät wird angesteckt.

Schritt 2: Der USB-Treiber des Thinclients bindet das Gerät ein.

Schritt 3: Der Thinclient leitet entsprechende Deskriptor-Felder oder die Kommunikationssignale des USB-Gerätes direkt an den Hypervisor weiter.

Schritt 4: Der Hypervisor prüft die Deskriptor-Felder gegen die Hypervisor-Richtlinie. Diese erlaubt entweder das Durchstellen oder verbietet es. Wird die Durchstellung verboten, wird das USB-Gerät nicht an die Hypervisor-Umgebung weitergeleitet und damit würde der Ablauf enden. Im Folgenden wird angenommen, dass das USB-Gerät weitergeleitet wird.

Schritt 5&6: Der Hypervisor fordert den USB-Gerät beim Thinclient an und bindet diesen ein.

Schritt 7: Der Hypervisor gibt das USB-Gerät an die virtuelle Maschine (VM) weiter.

Schritt 8: Die VM prüft das USB-Gerät anhand der Deskriptor-Felder gegen die VM-Richtlinie und bindet das Gerät entweder ein oder lehnt es ab.

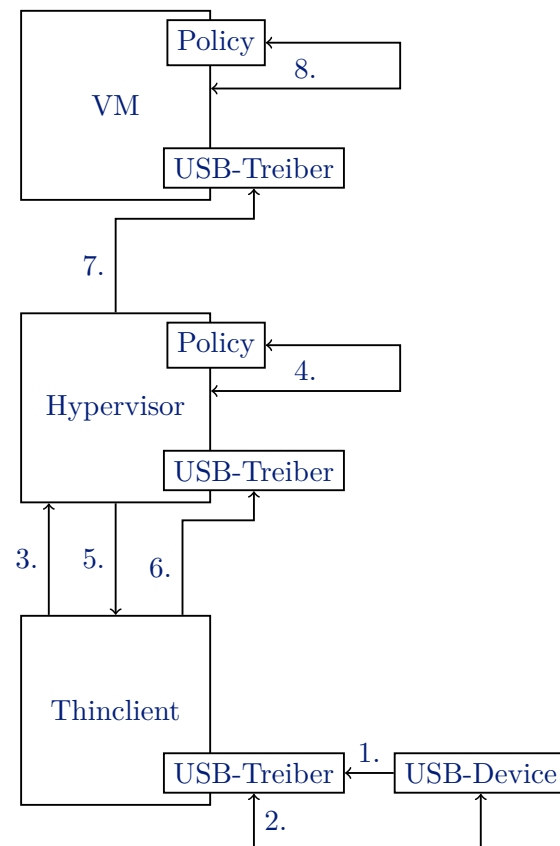


Abbildung 2.1.: Ablaufübersicht

3. Universal Serial Bus

Universal Serial Bus (USB) ist eine Schnittstelle, die so gut wie alle modernen Rechner besitzen. Es ist unter anderem möglich, darüber Geräte wie Kopfhörer und Joysticks aber auch Wechseldatenträger mit dem Rechner zu verbinden. Im letzteren Bereich ersetzt USB zunehmend die bisher vorherrschende CD/DVD-Technologie, da auf einem USB-Stick mehr Daten in höherer Geschwindigkeit gespeichert werden können und zudem die Geräte handlicher sind. Jedoch birgt die USB-Technologie auch Gefahren, die in den folgenden Sektionen beschrieben werden. Anschließend werden in 3.2 die Deskriptoren beschrieben, die ein USB-Gerät mit sich bringt und die für die Prüfung gegen die Richtlinie eine besondere Rolle spielen.

3.1. Gefahren bei USB

USB-Geräte stellen Gefahren auf verschiedenen Ebenen dar. Zum einen werden USB-Sticks, zumindest in dem Szenario, das hier betrachtet wurde, von Dritten an Mitarbeiter gegeben. Das bedeutet, dass ein Dritter, insofern er die nötige kriminelle Energie aufweist, ein präpariertes Gerät einschicken könnte. Erschwerend kommt hinzu, dass der Mitarbeiter keine Möglichkeit hat, ein schädliches USB-Gerät von einem normalen zu unterscheiden. Im folgenden werden einige der Risiken dargestellt.

3.1.1. Schadsoftware

Schadsoftware sind eine wachsende Bedrohung. Vor einigen Jahren waren nur einige wenige Virenfamilien weit verbreitet. So konnten Virenschutzhersteller über signaturbasierte Suchalgorithmen nach bekannten Mustern suchen und Viren zuverlässig identifizieren. In den letzten Jahren zeichnet sich jedoch der Trend ab, dass Viren sich schneller weiterentwickelt werden und zudem oft polymorph programmiert sind, also ihr Aussehen bei einer Infektion verändern. Dadurch werden signaturbasierte Erkennungen immer ineffektiver und die Gefahr, dass ein Rechner unerkannt infiziert wird, steigt. Eine Infektion passiert zumeist über sogenannte Browserexploits, also präparierte Webseiten, die Lücken in der Software des Users nutzen, oder Anhänge an Mails, die Schadcode enthalten. In dem hier betrachteten Szenario schickt ein krimineller oder aber auch ein unwissender Dritter, dessen Rechner im Vorfeld von einem Virus infiziert wurde, einen USB-Stick mit einem Schadsoftware ein. Ein Beispiel für einen solchen Virus wäre ein Trojaner. Diese tarnen sich als normale Software, beinhalten aber auch Schadcodefunktionalität. [12] Will ein Benutzer das vermeintlich sinnvolle Programm installieren, wird im Hintergrund unbemerkt die Schadroutine mitinstalliert und gestartet. Dieser Schadcode hat oftmals Funktionalitäten wie Keylogger, Backdoors oder ein Rootkit. Als Beispiel könnte man hier den in *Metasploit*[6] enthaltenen *Meterpreter* nennen, dessen Funktionen jedoch weit über die oben genannten hinaus gehen [7]. Hier muss also ein Benutzer einen USB-Stick einstecken und ein darauf befindliches Programm starten, damit sich

der Virus installieren kann. Ist er bereits bekannt, könnte ein auf dem System installierter Virens Scanner diesen erkennen und bestenfalls blockieren.

3.1.2. Datenabfluss

Neben den Gefahren von außen müssen auch sogenannte *Inside-Threats* beachtet werden. Dies wären Mitarbeiter, die z.B. interne IT-Systeme manipulieren, um sich Vorteile materieller oder immaterieller Art zu verschaffen. Bezogen auf USB wäre ein Risiko der Abfluss von vertraulichen oder wertvollen Daten, wenn also ein Mitarbeiter diese auf einem USB-Gerät speichert und aus dem Einflussbereich des Unternehmens bringt. Anschließend könnte er sich an diesen bereichern, falls es sich zum Beispiel um Bankdaten handelt. Andere denkbare Beispiele sind Kundendaten, Benutzerkonten, Geschäftsberichte vor der Veröffentlichung oder sonstige Unternehmensgeheimnisse. Diese können oft für Geld in einschlägigen Bereichen des Internets verkauft oder bei Geschäftsberichten zur Manipulation am Finanzmarkt genutzt werden. Auch wäre eine Abwerbung eines Mitarbeiters von einem anderen Unternehmen für Industriespionage denkbar. Eine weitere Bedrohung sind Geheimdienste, die Personen in ein Unternehmen einschleusen oder Mitarbeiter abwerben, um Daten über die Kunden zu sammeln. Dies wurde erst kürzlich durch von Edward Snowden veröffentlichte Dokumente publik.[11]

3.1.3. Exploits auf Treiberebene

Für den Mitarbeiter noch schwieriger zu entdecken sind Exploits auf Treiberebene. Dieses Vorgehen ist relativ neu. Es werden dabei Lücken im Treiber des Geräts ausgenutzt, um Schadcode auszuführen. Hierzu muss ein Benutzer einen manipulierten USB-Stick nur einstecken. Es bedarf im Gegensatz zu normalen Viren keiner weiteren Interaktion des Users, da der Computer automatisch mit dem USB-Device kommuniziert, um die Funktionen des Geräts zu erfahren und eventuell benötigte Treiber zu installieren. Hier beginnt das Gerät jedoch bereits, bestimmte schädliche Zeichenfolgen an den Computer zu senden, die vom Treiber interpretiert werden und unter Umständen einen Buffer Overflow oder eine andere Schwachstelle ausnutzen können[2]. Durch diese Lücken kann dann auf dem Rechner des Benutzers unbemerkt Schadcode ausgeführt werden.

3.2. Deskriptoren

Die USB-Spezifikation, die von dem USB Implementers Forum, Inc.[14] festgelegt wird, sieht folgende Felder vor, die Informationen zu dem Gerät beinhalten.

Bytes		
1	bNumConfigurations	Dies umfasst die technischen Informationen, wie die Länge der gesamten Felder im <i>bLength</i> oder das Protokoll des Geräts im <i>bDeviceProtocol</i> über Informationen für das Betriebssystem wie <i>idVendor</i> , <i>idProduct</i> , <i>bDeviceSubClass</i> und <i>bDeviceClass</i> . Diese Felder mit der jeweiligen Länge sind in der Grafik dargestellt. Ein Feld hat dabei zwischen ein und zwei Bytes. Die Felder <i>bDeviceClass</i> , <i>bDeviceSubClass</i> , <i>bDeviceProtocol</i> sowie <i>idVendor</i> werden vom Hersteller befüllt.[13] Das Betriebssystem nutzt die Felder meist, um Treiber zu suchen oder auch das angeschlossene USB-Gerät gegen die Policy-Einstellungen zu prüfen. Um eigene Werte bei <i>idProduct</i> oder <i>idVendor</i> -Felder zu nutzen und damit sicher gestellt ist, dass nicht mehrere Hersteller dieselbe <i>idProduct</i> verwenden, müssen die Adressbereiche der <i>idProduct</i> bei dem USB Implementers Forum, Inc. gekauft werden. Dazu gibt es zwei Möglichkeiten. Man kann entweder ein Mitglied des Forums werden oder für einen einmaligen Betrag einen Adressraum erstehen. Im zweiten Fall darf man jedoch nicht das offizielle USB-Logo verwenden. [15] Im folgenden werden die für dieses Dokument interessanten Felder weiter erläutert:
1	iSerialNumber	
1	iProduct	
1	iManufacturer	
2	bcdDevice	
2	idProduct	
2	idVendor	
1	bMaxPacketSize	
1	bDeviceProtocol	
1	bDeviceSubClass	
1	bDeviceClass	
2	bcdUSB	
1	bDescriptorType	
1	bLength	

idVendor: Der Inhalt des *idVendor*-Feld wird von der USB Implementers Forum, Inc. festgelegt. Das Feld ist 2 Byte lang und jeder Wert ist genau einem Hersteller zugeordnet. Erstet ein Unternehmen einen *idVendor*-Wert, kann er, solange er diesen *idVendor*-Wert nutzt, frei über das *idProduct*-Feld verfügen.

idProduct: Das *idProduct*-Feld wird von dem Unternehmen vergeben, das einen Wert im *idVendor*-Feld gekauft hatte. Es ist ebenfalls 2 Bytes lang. Damit kann ein Unternehmen bis zu 2^{16} verschiedene Produkte beschreiben.

bInterfaceClass & bInterfaceSubClass & bInterfaceProtocol: Diese Felder sind in der oberen Aufstellung nicht enthalten, da sie nicht über das Device, sondern über das *USB-Interface* Auskunft geben. Sie beschreiben die Klasse, die Unterklasse und das Protokoll, über das mit dem Interface kommuniziert werden soll.

4. Richtlinien

Eine Policy ist ein Regelwerk, das die Rechte und Möglichkeiten von Benutzern auf einem IT-System beschreibt und eingrenzt. Es gibt verschiedene Arten von Policies. Im folgenden wird nur die beschrieben, die für den weiteren Verlauf der Arbeit relevant ist. Hier besteht eine Regel aus mehreren Bestandteilen, die entweder wahr oder falsch sind. Diese Bestandteile können per *und*- oder *oder*-Verknüpfungen zu einer Regel vereint werden.

Abstrakt ist eine Policy mit einem Regelwerk wie der Straßenverkehrsordnung zu vergleichen. Auch hier gibt es Vorgaben, wer, wann und wo fahren oder parken darf. So wird zum Beispiel bei einem Durchfahrtsverbot, das für Anlieger nicht gelten soll, folgende Regel festgelegt:

Regel-1: Die Durchfahrt ist für alle verboten

Regel-2: *oder* der Fahrer ist Anlieger.

Bezeichnen wir in dem Beispiel den Ausgang *der Fahrer darf durch die Straße fahren* als 1 und den Ausgang *der Fahrer darf nicht durch die Straße fahren*, als 0, so wäre hier das Ergebnis

$$\alpha = \text{Regel-1} \vee \text{Regel-2}$$

mit

$$\text{Regel-1} = 0$$

. Somit ergeben sich daraus für die verschiedenen Fälle von *Regel-2*

$$\alpha = \begin{cases} 0 & \text{wenn Regel-1 gleich 0} \\ 1 & \text{wenn Regel-1 gleich 1} \end{cases}$$

. Ähnliche Regeln können über Policies auf Rechner festgelegt werden. Hier wäre eine mögliche vergleichbare Regel im Bezug auf Speicherzugriffe

1. Der Zugriff auf diesen Ordner ist gesperrt
2. außer der Benutzer hat die Kennung MaxMuster

Solche Regeln werden jedoch nicht nur für die Organisation von Speicherzugriffen, sondern auch für das Sperren bestimmter Einstellungen oder mancher Geräte verwendet.

5. Umgehung der USB-Richtlinie

5.1. Wie wird gefiltert?

Die in diesem Dokument benutzten USB-Policies werden über die in Abschnitt 3.2 beschriebenen USB-Deskriptoren definiert. Wollten wir etwa ein Gerät mit *idProduct* = 0x01 und *idVendor* = 0x02 freigeben, aber alle sonstigen Geräte abweisen, so wäre folgende Regel möglich:

- Verbiete alle USB-Geräte *oder*
- Erlaube USB-Geräte mit
 - *idProduct* = 0x01 *und*
 - *idVendor* = 0x02

Die Regeln werden von oben nach unten gelesen, wobei spätere Regeln frühere überschreiben. Hier würden also zuerst alle USB-Geräte blockiert, außer das Gerät besitzt die *idProduct* = 0x01 und *idVendor* = 0x02. Dies scheint logisch. Hat ein Gerät z.B. die *idProduct* = 0x03, so tritt die *Erlaube*-Regel nicht in Kraft und es bleibt die *Verbiete*-Regel bestehen. Meldet sich ein Gerät mit *idProduct* = 0x01 und *idVendor* = 0x02 an, so gilt zwar auch zunächst die *Verbiete*-Regel, jedoch trifft die *Erlaube*-Regel zu und überschreibt die *Verbiete*-Regel, sodass der Zugriff gewährt wird. Diese Zugriffe können gegebenenfalls noch um eine *Active-Directory-Gruppe* erweitert werden. Dies ist vor allem nützlich, wenn man nur bestimmten Benutzern die Möglichkeit geben will, USB-Geräte zu nutzen. Wollten wir z.B. dem Benutzer „Alice“ den Zugriff auf ein USB-Gerät mit der *idProduct* = 0x01 und der *idVendor* = 0x02 geben, so wäre die Regel:

- Verbiete alle USB-Geräte *oder*
- Ist *User* = *Alice*
- Erlaube USB-Geräte mit
 - *idProduct* = 0x01
 - *idVendor* = 0x02

5.2. Teensy

Das Teensy ist eine Platine, bestehend aus einem 72 MHz MK20DX256VLH7 Cortex-M4 Prozessor, 256 kbytes Flash Speicher und 64 kbytes RAM. Zudem verfügt es über eine USB-Schnittstelle. Man kann also ein Programm auf dem Teensy ablegen, das ausgeführt wird, wenn man den USB-Stick einsteckt. So kann man beliebige Signalfolgen über die USB-Schnittstelle an ein anderes Gerät schicken.

5.3. Konzept

Die USB-Felder sind allgemein nicht durch Signaturen oder sonstige Möglichkeiten vor Manipulation geschützt sind. Es soll untersucht werden, ob es möglich ist, einen Teensy so zu programmieren, dass er sich als ein beliebiges Gerät ausgibt, also beliebige *idProduct*- und *idVendor*-Werte emuliert. Beschränkt eine USB-Policy den Zugriff auf ein bestimmtes Gerät, so könnte man dieses theoretisch mit dem Teensy nachahmen. Um dies zu untersuchen, wurden verwendet:

- Teensy 3.1 + USB-Kabel
- Die Entwicklungsumgebung Arduino 1.0.5 (64bit) installiert unter `~/teensy/arduino-1.0.5`
- Die Erweiterung Teensyduino 1.19 (64bit)
- Kali-Linux als Testbetriebssystem (64bit)

Zur Analyse, mit welchen *idVendor* und *idProduct*-Werten sich der Teensy meldet, wurde mit dem Kommando „`tail -f /var/log/syslog`“ das zentrale Logfile des Linuxsystems ausgelesen. Beim ersten Einstecken ergab sich dabei folgende Meldung:

```
[...]
Nov  8 14:26:50 kali kernel: [ 9438.405774] usb 1-2: new full-speed USB device number 11 using xhci_hcd
Nov  8 14:26:50 kali kernel: [ 9438.534904] usb 1-2: New USB device found, idVendor=16c0, idProduct=0483
Nov  8 14:26:50 kali kernel: [ 9438.534913] usb 1-2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
Nov  8 14:26:50 kali kernel: [ 9438.534918] usb 1-2: Product: USB Serial
Nov  8 14:26:50 kali kernel: [ 9438.534922] usb 1-2: Manufacturer: Teensyduino
Nov  8 14:26:50 kali kernel: [ 9438.534925] usb 1-2: SerialNumber: 413450
Nov  8 14:26:50 kali kernel: [ 9438.535640] cdc_acm 1-2:1.0: This device cannot do calls on its own. It is not a
modem.
Nov  8 14:26:50 kali kernel: [ 9438.535680] cdc_acm 1-2:1.0: ttyACM0: USB ACM device
Nov  8 14:26:50 kali mtp-probe: checking bus 1, device 11: "/sys/devices/pci0000:00/0000:00:14.0/usb1/1-2"
Nov  8 14:26:50 kali mtp-probe: bus: 1, device: 11 was not an MTP device
Nov  8 14:26:56 kali kernel: [ 9444.328208] usb 1-2: USB disconnect, device number 11
[...]
```

Listing 5.1: tail -f syslog output

Die wichtigen Werte, also die *idVendor* gleich *16c0* und die *idProduct* gleich *0483* sind farblich hervorgehoben und unterstrichen. Die *bInterface*-Felder sind jedoch aus dem *syslog* nicht erkenntlich. Hierzu wird der *lsusb*-Befehl verwendet. Die Ausgabe sieht wie folgt aus:

```
Bus 001 Device 009: ID 16c0:0483 V0TI Teensyduino Serial
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB                  2.00
  bDeviceClass             2 Communications
  bDeviceSubClass          0
  bDeviceProtocol          0
  bMaxPacketSize0         64
  idVendor                0x16c0 V0TI
  idProduct               0x0483 Teensyduino Serial
  bcdDevice                1.00
  iManufacturer           1 Teensyduino
  iProduct                 2 USB Serial
  iSerial                  3 413450
  bNumConfigurations       1
Configuration Descriptor:
  bLength                 9
  bDescriptorType         2
  wTotalLength             67
  bNumInterfaces          2
  bConfigurationValue      1
  iConfiguration           0
  bmAttributes             0xc0
    Self Powered
```

```
MaxPower          100mA
Interface Descriptor:
  bLength          9
  bDescriptorType  4
  bInterfaceNumber 0
  bAlternateSetting 0
  bNumEndpoints    1
  bInterfaceClass   2 Communications
  bInterfaceSubClass 2 Abstract (modem)
  bInterfaceProtocol 1 AT-commands (v.25ter)
```

Listing 5.2: `lsusb -v -d16c0:0483`

Der `lsusb`-Befehl zeigt hier bei dem Gerät, als welches sich der Teensy ausgeben soll, folgende Ausgabe:

```
Bus 007 Device 030: ID 0e8d:1887 MediaTek Inc.
Device Descriptor:
  bLength        18
  bDescriptorType 1
  bcdUSB         2.00
  bDeviceClass    0 (Defined at Interface level)
  bDeviceSubClass 0
  bDeviceProtocol 0
  bMaxPacketSize0 64
  idVendor       0x0e8d MediaTek Inc.
  idProduct      0x1887
  bcdDevice      0.00
  iManufacturer  1 MediaTek Inc
  iProduct       2 MT1887
  iSerial        3 KZ5D5K91838
  bNumConfigurations 1
Configuration Descriptor:
  bLength        9
  bDescriptorType 2
  wTotalLength   32
  bNumInterfaces 1
  bConfigurationValue 1
  iConfiguration 4 Default
  bmAttributes   0xa0
    (Bus Powered)
    Remote Wakeup
  MaxPower       500mA
Interface Descriptor:
  bLength        9
  bDescriptorType 4
  bInterfaceNumber 0
  bAlternateSetting 0
  bNumEndpoints  2
  bInterfaceClass  8 Mass Storage
  bInterfaceSubClass 2 SFF-8020i, MMC-2 (ATAPI)
  bInterfaceProtocol 80
```

Listing 5.3: `lsusb -v -0e8d:1887`

Nun bearbeitet man die unter `arduino-1.0.5/hardware/teensy/cor-es/teensy3/` liegende `usb_desc.h` sowie die `usb_desc.c`, welche die notwendigen Informationen bei einer Neubeschreibung des Teensy bereit halten. Die relevanten Abschnitte sowie die zu ändernden Werte sind wieder farblich hinterlegt und unterstrichen.

```
#if defined(USB_SERIAL)
#define VENDOR_ID          0x0e8d          //0x152d
#define PRODUCT_ID       0x1887          //0x2339
#define DEVICE_CLASS     0x00           //war 2
#define DEVICE_SUBCLASS  0x00           //Neu Eingefuegt
#define DEVICE_PROTOCOL  0x00           //Neu Eingefuegt
#define MANUFACTURER_NAME {'T','e','n','s','y','d','u','i','n','o'}
#define MANUFACTURER_NAME_LEN 11
#define PRODUCT_NAME      {'U','S','B',' ','S','e','r','i','a','l'}
#define PRODUCT_NAME_LEN  10
#define EPO_SIZE          64
#define NUM_ENDPOINTS     4
#define NUM_USB_BUFFERS   12
#define NUM_INTERFACE     2 //was 2
```

Listing 5.4: Ausschnitt: `usb_desc.h`, Zeile 87 bis 100

```
#ifndef CDC_DATA_INTERFACE
// interface descriptor, USB spec 9.6.5, page 267-269, Table 9-12
9, // bLength
```

```

4,                                // bDescriptorType
CDC_STATUS_INTERFACE,           // bInterfaceNumber
0,                                // bAlternateSetting
1,                                // bNumEndpoints
0x08,                            // bInterfaceClass, war 2
0x02,                            // bInterfaceSubClass, war 2
0x50,                            // bInterfaceProtocol, war 1
5,                                // iInterface was 0

```

Listing 5.5: Ausschnitt: usb_desc.c, Zeile 334 bis 344

Ändert man hier die markierten Werte und beschreibt den Teensy mittels der Arduino-Software neu, so werden diese Deskriptoren verwendet. Die Einstellungen hierfür können aus der Grafik A.1 im Anhang entnommen werden. Das Programm ist dabei entbehrlich, hier wurde eine an den Teensy 3.1 angepasste Version des *Blink*-Programms verwendet, welches im Anhang abgelegt ist. Kompiliert man das Programm nun und lädt es auf den Teensy, ergibt der „tail -f /var/log/syslog“-Befehl folgenden Output

```

[...]
Nov 8 15:05:03 kali dbus[2582]: [system] Failed to activate service 'org.freedesktop.Avahi': timed out
Nov 8 15:05:17 kali kernel: [11746.713963] usb 1-1: new full-speed USB device number 15 using xhci_hcd
Nov 8 15:05:17 kali kernel: [11746.842959] usb 1-1: New USB device found, idVendor=0e8d, idProduct=1887
Nov 8 15:05:17 kali kernel: [11746.842968] usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
Nov 8 15:05:17 kali kernel: [11746.842973] usb 1-1: Product: USB Serial
Nov 8 15:05:17 kali kernel: [11746.842976] usb 1-1: Manufacturer: Teensyduino
Nov 8 15:05:17 kali kernel: [11746.842980] usb 1-1: SerialNumber: 413450
Nov 8 15:05:17 kali kernel: [11746.843679] usb-storage 1-1:1.0: USB Mass Storage device detected
Nov 8 15:05:17 kali kernel: [11746.844125] usb-storage: probe of 1-1:1.0 failed with error -5
[...]

```

Listing 5.6: tail -f syslog output 2

und *lsusb* meldet:

```

Bus 001 Device 017: ID 0e8d:1887 MediaTek Inc.
Device Descriptor:
  bLength                18
  bDescriptorType        1
  bcdUSB                 2.00
  bDeviceClass            0 (Defined at Interface level)
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0        64
  idVendor              0x0e8d  MediaTek Inc.
  idProduct             0x1887
  bcdDevice              1.00
  iManufacturer          1  Teensyduino
  iProduct                2  USB Serial
  iSerial                 3  413450
  bNumConfigurations     1
Configuration Descriptor:
  bLength                9
  bDescriptorType        2
  wTotalLength           67
  bNumInterfaces         2
  bConfigurationValue    1
  iConfiguration         0
  bmAttributes           0xc0
    Self Powered
  MaxPower               100mA
Interface Descriptor:
  bLength                9
  bDescriptorType        4
  bInterfaceNumber       0
  bAlternateSetting      0
  bNumEndpoints          1
  bInterfaceClass       8  Mass Storage
  bInterfaceSubClass    2  SFF-8020i, MMC-2 (ATAPI)
  bInterfaceProtocol    80

```

Listing 5.7: lsusb -v -0e8d:1887

Wie man den hervorgehobenen Stellen sehen kann, meldet sich der Teensy nun mit geänderten Deskriptoren. Das *iManufacturer*-Feld wird bei der Filterung der Policy nicht genutzt und wurde daher zu veranschaulichungszwecken nicht geändert.

5.4. Proof of Concept

Die Policy der virtuellen Umgebung ist so eingestellt, dass nur eine bestimmte Baureihe eines USB-Laufwerks an die virtuelle Umgebung durchgestellt wird.

```
[...]  
07:00:43 VUSB: Redirected Device(0x0e8d,01887,0x08,0x02,0x50)  
07:00:43 USB: Found new device "WYSE VUSB" on port 2  
07:00:44 CTX USB: redirected device 5 (PID: 0x1887, VID 0x0e8d)  
[...]
```

Listing 5.8: Zugelassenes CD-Laufwerk

Wird versucht, ein Gerät mit abweichenden Deskriptor-Werten zu verbinden, so wird diese nicht durchgestellt.

```
[...]  
07:09:30 VUSB: Local Device(0x152d,0x2339,0x02,0x00,0x00)  
07:09:30 USB: Found new device "ACM momdem" on port 2  
[...]
```

Listing 5.9: Standard Teensy

Werden die Deskriptoren des Teensies auf die des USB-Laufwerks geändert und angesteckt, so wird das Gerät durchgestellt.

```
[...]  
07:48:39 VUSB: Redirected Device(0x0e8d,01887,0x08,0x02,0x50)  
07:48:39 USB: Found new device "WYSE VUSB" on port 2  
07:48:40 CTX USB: redirected device 5 (PID: 0x1887, VID 0x0e8d)  
[...]
```

Listing 5.10: Teensy, der sich als das CD-Laufwerk ausgibt

6. Fazit und Gegenmaßnahmen

Da der USB-Standard keine Möglichkeit bietet, Geräte fehlerfrei zu identifizieren, zum Beispiel über Signaturen, kann man diese Ebene nicht als effektive Schutzmaßnahme einstufen und muss die Gefahren direkt eindämmen. Jedoch ist dies im Bezug auf die Exploits auf Treiberebene sehr schwer. Die einfachste und sicherste Methode wäre, die Benutzung von USB-Ports in einem Unternehmen per Richtlinie zu verbieten und die Ports eventuell sogar noch physikalisch zu versiegeln. Hier hätte man natürlich den Nachteil, dass USB-Geräte nicht mehr direkt genutzt werden könnten. Als Lösung für dieses Problem könnte eine Art Schleusen-System für USB-Geräte aufgebaut werden. So könnte, wenn ein USB-Stick an die Firma geschickt wird, dieser zentral von einer IT-Stelle in einen Terminal-Server eingebunden und die Daten an den gewünschten Empfänger weiterreicht werden. Die Vorteile sind hier, dass, falls Viren auf dem USB-Stick enthalten sind, diese vorher am Terminal-Server sowie bei der Netzwerkübertragung von mehreren verschiedenen Virens Scanner überprüft sowie heuristischen Analysen unterworfen werden könnten. Ebenso würde bei einem manipulierten USB-Stick nicht der Rechner des Mitarbeiters, sondern nur der Schleusen-PC infiziert. Trifft man hier entsprechende Sicherheitsmaßnahmen, wie die Platzierung des Schleusen-PCs in der demilitarisierten Zone und einen regelmäßigen Tausch oder Neuinstallation des Schleusenrechners, kann man das Risiko minimieren. Der Mitarbeiter würde in diesem Fall nur die Dateien bekommen und wäre von der Manipulation auf Treiberebene nicht betroffen. Zudem sind die im Vorfeld getroffenen Sicherheitsmaßnahmen für den Mitarbeiter transparent.

A. Appendix

A.1. Quellcode

Listing A.1: Blink_2.ino

```
1  /* LED Blink, Teensyduino Tutorial #1
2     http://www.pjrc.com/teensy/tutorial.html
3
4     This example code is in the public domain.
5  */
6
7  // Teensy 2.0 has the LED on pin 11
8  // Teensy++ 2.0 has the LED on pin 6
9  // Teensy 3.0 has the LED on pin 13
10 const int ledPin = 13;
11
12 // the setup() method runs once, when the sketch starts
13
14 void setup() {
15     // initialize the digital pin as an output.
16     pinMode(ledPin, OUTPUT);
17 }
18
19 // the loop() method runs over and over again,
20 // as long as the board has power
21
22 void loop() {
23     digitalWrite(ledPin, HIGH);    // set the LED on
24     delay(1000);                   // wait for a second
25     digitalWrite(ledPin, LOW);     // set the LED off
26     delay(1000);                   // wait for a second
27 }
```


A.2. Ergänzende Grafiken

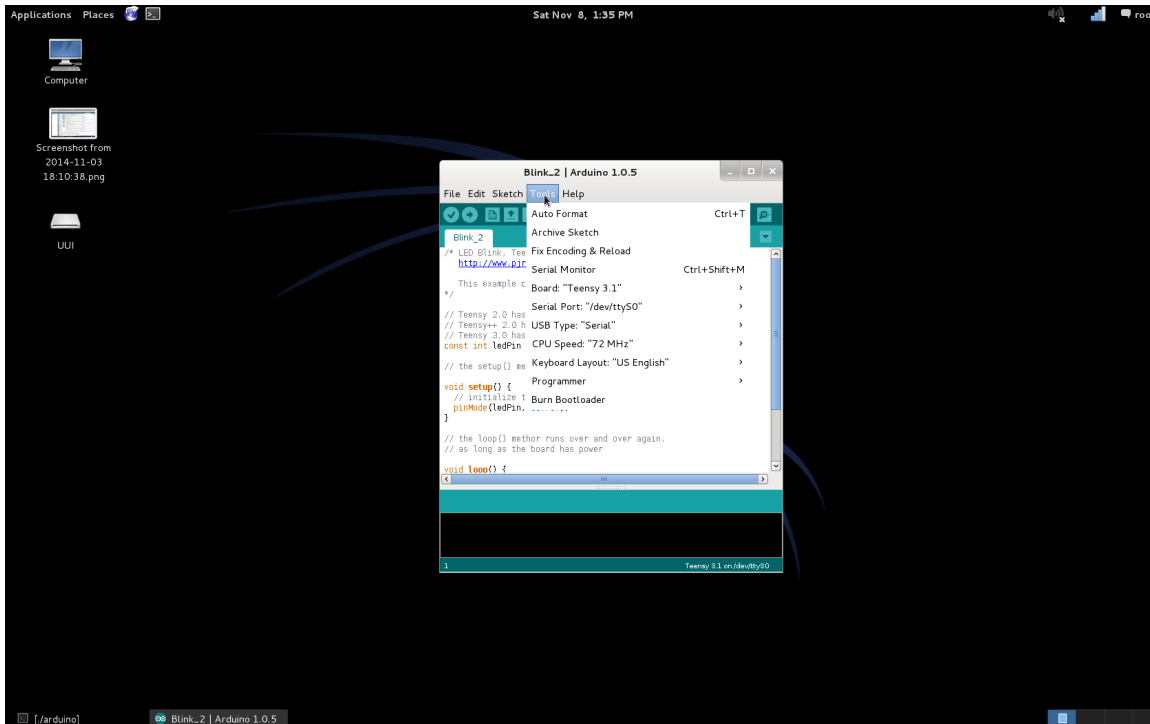


Abbildung A.1.: Einstellungen Arduino

A.3. Quellcode Grafiken

Listing A.2: Ablaufdiagramm 2.1

```

1 \begin{figure}[htbp]
2 \centering
3 \begin{tikzpicture}[
4     scale=1,
5     line width=0.25mm,
6     every node/.style={
7         scale=1,
8         text=THIblue},
9     align=center,
10    node distance=4cm,
11    comp/.style={
12        fill=white,
13        rectangle,
14        draw,
15        minimum size=2.5cm},
16    driver/.style={
17        fill=white,

```

```
18         rectangle ,
19         draw ,
20         yshift=2cm,
21         xshift=-1cm},
22 device/.style={
23     fill=white,
24     rectangle,
25     draw},
26 policie/.style={
27     fill=white,
28     rectangle,
29     draw,
30     yshift=-2cm,
31     xshift=-1.5cm}
32 ]
33
34 \node[comp] (thinclient) at (0,0){Thinclient};
35 \node[driver] (thinclientUSB) [below right of=thinclient] {
36     USB-Treiber};
37 \node[device] (USBDevice) [right of=thinclientUSB, xshift=-1
38     cm] {USB-Device};
39
40 \node[comp] (hypervisor) [above of=thinclient] {Hypervisor};
41 \node[driver] (hypervisorUSB) [below right of=hypervisor] {
42     USB-Treiber};
43 \node[policie] (hypervisorPol) [above right of=hypervisor] {
44     Policy};
45
46 \node[comp] (VM) [above of=hypervisor] {VM};
47 \node[driver] (VMUSB) [below right of=VM] {USB-Treiber};
48 \node[policie] (VMPol) [above right of=VM] {Policy};
49
50 \draw[->]
51     (USBDevice) --
52         node[above]{1.}
53         (thinclientUSB);
54
55 \draw[<->]
56     (thinclientUSB) --
57         node[right]{2.}
58         ++(0,-1) -| (USBDevice);
59
60 \draw[->]
61     (thinclient.125) --
62         node[left]{3.}
63         (hypervisor.235);
64
65 \draw[<->]
66     (hypervisor.10) --
67         node[above]{4.}
68         ++(2.5,0) |- (hypervisorPol);
```

```

66 \draw[->]
67     (hypervisor.270) --
68         node[left]{5.}
69         (thinclient.90);
70
71 \draw[->]
72     (thinclient.55) --
73         node[left]{6.}
74         ++(0,1.25) -| (hypervisorUSB);
75
76 \draw[->]
77     (hypervisor) --
78         node[left]{7.}
79         ++(0,2.5) -| (VMUSB);
80
81 \draw[<->]
82     (VM.10) --
83         node[above]{8.}
84         ++(2.5,0) |- (VMPol);
85
86 \end{tikzpicture}
87 \caption{Ablaufübersicht}
88 \label{fig:Ablauf}
89 \end{figure}

```

Listing A.3: USB Deskriptoren 3.2

```

1 \begin{wrapfigure}{l}{0pt}
2 \begin{tikzpicture}[scale=1, text=THIblue]
3     \draw (0,0) rectangle (0.5,0.5);
4     \draw (0.25, 0.25) node {1};
5     \draw (0.5, 0.25) node[right]{bLength};
6
7     \draw (0,0.5) rectangle (0.5,0.5);
8     \draw (0.25, 0.75) node {1};
9     \draw (0.5, 0.75) node[right]{bDescriptorType};
10
11     \draw (0,1) rectangle (0.5,1);
12     \draw (0.25,1.5) node {2};
13     \draw (0.5,1.5) node[right]{bcdUSB};
14
15     \draw (0,2) rectangle (0.5,0.5);
16     \draw (0.25,2.25) node {1};
17     \draw (0.5,2.25) node[right]{bDeviceClass};
18
19     \draw (0,2.5) rectangle (0.5,0.5);
20     \draw (0.25,2.75) node {1};
21     \draw (0.5,2.75) node[right]{bDeviceSubClass};
22
23     \draw (0,3) rectangle (0.5,0.5);
24     \draw (0.25,3.25) node {1};
25     \draw (0.5,3.25) node[right]{bDeviceProtocol};

```

```
26
27     \draw (0,3.5) rectangle (0.5,0.5);
28     \draw (0.25,3.75) node {1};
29     \draw (0.5,3.75) node[right]{bMaxPacketSize};
30
31     \draw (0,4) rectangle (0.5,1);
32     \draw (0.25,4.5) node {2};
33     \draw (0.5,4.5) node[right]{idVendor};
34
35     \draw (0,5) rectangle (0.5,1);
36     \draw (0.25,5.5) node {2};
37     \draw (0.5,5.5) node[right]{idProduct};
38
39     \draw (0,6) rectangle (0.5,1);
40     \draw (0.25,6.5) node {2};
41     \draw (0.5,6.5) node[right]{bcdDevice};
42
43     \draw (0,7) rectangle (0.5,0.5);
44     \draw (0.25,7.25) node {1};
45     \draw (0.5,7.25) node[right]{iManufacturer};
46
47     \draw (0,7.5) rectangle (0.5,0.5);
48     \draw (0.25,7.75) node {1};
49     \draw (0.5,7.75) node[right]{iProduct};
50
51     \draw (0,8) rectangle (0.5,0.5);
52     \draw (0.25,8.25) node {1};
53     \draw (0.5,8.25) node[right]{iSerialNumber};
54
55
56     \draw (0,8.5) rectangle (0.5,0.5);
57     \draw (0.25,8.75) node {1};
58     \draw (0.5,8.75) node[right]{bNumConfigurations};
59
60     \draw (0,9) rectangle (0.5,0.5);
61
62     \draw (0.25, 9) node[rotate=90, right] {Bytes};
63 \end{tikzpicture}
64 \label{fig:usbDeskriptoren}
65 \end{wrapfigure}
```

Literatur

- [1] *AB Acquisition LLC Confirms Incident Involving Payment Card Data Processing.* URL: <http://www.jewelosco.com/2014/08/ab-acquisition-llc-confirms-incident-involving-payment-card-data-processing/> (besucht am 01.11.2014).
- [2] *Bad USB.* URL: <https://srlabs.de/badusb/> (besucht am 03.11.2014).
- [3] *Heartbleed.* URL: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160> (besucht am 29.10.2014).
- [4] *Home Depot probes possible hack, could be larger than Target breach.* URL: http://www.denverpost.com/business/ci_26453916/data-stolen-from-11-colorado-goodwill-stores-home (besucht am 01.11.2014).
- [5] *McAfee.* URL: http://csis.org/files/attachments/140609_McAfee_PDF.pdf (besucht am 29.10.2014).
- [6] *Metasploit.* URL: <http://www.metasploit.com/> (besucht am 03.11.2014).
- [7] *Meterpreter.* URL: http://www.offensive-security.com/metasploit-unleashed/Meterpreter_Basics (besucht am 03.11.2014).
- [8] *Poodle.* URL: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3566> (besucht am 08.11.2014).
- [9] *Shellshock.* URL: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271> (besucht am 29.10.2014).
- [10] *Snowden.* URL: <https://www.theguardian.com/us-news/edward-snowden> (besucht am 29.10.2014).
- [11] *Spionageaktivitäten auf Unternehmen.* URL: <https://firstlook.org/theintercept/2014/10/10/core-secrets/> (besucht am 03.11.2014).
- [12] Mark Stamp. *Information Security: Principles And Practice.* S. 281 Malware. John Wiley & Sons, Inc., Hoboken, New Jersey, 2006.
- [13] *USB 2.0 Standart.* URL: http://www.usb.org/developers/docs/usb20_docs/#usb20spec (besucht am 29.10.2014).
- [14] *USB Implementers Forum, Inc.* URL: <http://www.usb.org/about> (besucht am 29.10.2014).
- [15] *USB Vendor.* URL: <http://www.usb.org/developers/vendor/> (besucht am 29.10.2014).